Supporting Temporal Queries on Clinical Relational Databases: The S-WATCH-QL Language

Carlo Combi (§), Luca Missora and Francesco Pinciroli (°*)

§ Dipartimento di Matematica e Informatica, Universita' degli Studi di Udine
° Dipartimento di Bioingegneria del Politecnico di Milano
* Centro di Teoria dei Sistemi del CNR, Milano

Due to the ubiquitous and special nature of time, specially in clinical databases there's the need of particular temporal data and operators. In this paper we describe S-WATCH-QL (Structured Watch Query Language), a temporal extension of SQL, the widespread query language based on the relational model. S-WATCH-QL extends the well-known SQL by the addition of: a) temporal data types that allow the storage of information with different levels of granularity; b) historical relations that can store together both instantaneous valid times and intervals; c) some temporal clauses, functions and predicates allowing to define complex temporal queries.

INTRODUCTION

In medical informatics, the temporal dimension is very important; it is basic either in the decision making activity in diagnosis, therapy and prognosis, or in the analysis of clinical data for scientific purposes.[1, 2] Clinical information consists both of natural language sentences (such as anamneses, therapies), and of quantitative parameters (heart rate, systolic blood pressure). Temporal localization of such data may be expressed by using different time units. Granularity of temporal information is, in fact, the accuracy or the unit of measure used for the temporal axis (e.g. days, hours, minutes).[2, 3] Some attempts have been made to manage and store temporal clinical information with different granularity at the database level.[3, 4, 5, 6] Besides the need of using different time units, there are also many other needs related to the temporal management of clinical data. More precisely, in defining and implementing S-WATCH-QL (Structured Watch Query Language), we faced the following problems in managing temporal clinical information.

a) compatibility with the flat relational model and with SQL[7]; a large amount of clinical data is stored in conventional relational databases. Temporal queries on these data have to be performed. Temporal query languages have to consider also flat data, containing some user-defined temporal dimension.

b) global management of instant- and interval- valid times with different granularities; a tuple's valid time represents the time during which the information contained in the tuple is considered to be valid in the modeled real-world.[8] Usually in a relation valid times are homogeneous both in granularity and in instant/interval reference.[8, 5] This is a limitation in the clinical domain: in a clinical relation containing descriptions of pathologies, for example, instantaneous tuples (e.g. "myocardial infarction at 21:12 of March 16th, 1994") and interval-based tuples (e.g. "atrial fibrillation on February 12th 1995 from 18:30:15 to 18:45:34") must co-exist.

c) addition and enhancement of some specific clauses, functions and predicates. Some capabilities are required to consider temporal clinical data about features like temporal proximity, temporal order, patient state identification related to complex temporal relationships between collected data. For example, it is important the capability to observe data by a window, having a predefined duration, moving on the time axis (e.g. "find the patients having measures of Hearth Rate below 70 bpm for ten days").

In the following we describe the main features of the language S-WATCH-QL and also show an application of the language to the management of data coming from an anesthesia record.[10]

THE HISTORICAL RELATIONAL DATA MODEL

Basic concepts

The relational model is the most important model[9], used in commercial database systems. *A relational database* consists of a set of *relations*, each of which may be represented as a table. In a table a row (called *tuple*) represents a relation among a set of values. All the values belonging to a given column are associated with an *attribute*, which is made up of a *name* that identifies the attribute and a *domain*, that specifies the set of valid values. Recently many researchers extended the relational model including temporal information and operators that manage it.[11]

Our extension can be classified as a tuple time-stamped historical relational data model. An historical database is composed of *non-historical relations* and *historical relations*. Non-historical relations are traditional relations with no time support (other than user defined time). Historical relations contain an additional attribute, named TIME, whose value, for each tuple, defines the *valid time* of the values of the

0195—4210/96/$5.00

other attributes. A tuple's valid time represents the time during which the information contained in the tuple is considered to be valid.[8]
An *instant* defines a position on the temporal axis. It can be defined by values specified with different granularities (years, months, days,...).
An instant has no duration; an instant defined with a certain level of granularity has an associated indeterminacy. *Intervals* are defined by couples of instants that indicate their start and end. Start and end can be defined at different levels of granularity: an interval is, for example, "from 1990 to April 1996" (1990 .. 1996:4 in the adopted notation). Intervals are closed; this means that start and end are included: the interval 1996:1:15..1996:1:18 includes both January, 15 and January, 18. An interval defined with equal start and end, like 1996:1 .. 1996:1 indicates an interval, lasting in this case the whole month of January. By using a special operator it is possible to assign to an interval type variable *i*, a value made up of two equal instants (by either value or granularity level), that represents an interval with zero duration (in contrast with the default behavior of closed intervals, with starting and ending instants having the same value, that represent a given non zero duration). We call this a *degenerate interval.*[9]
Traditionally there are two ways to define the valid time of an historical relation: as a time *instant* (a position on the time axis) and as a time *interval*, a set of contiguous instants.[11] Generally a historical relation can contain valid times that are all instants or all intervals.[5, 9] In our model, instead, the attribute TIME (that defines a tuple's valid time) is of type interval, then it works good with interval valid times; moreover, by defining the attribute TIME as a degenerate interval, it is possible to represent instantaneous valid times.

THE QUERY LANGUAGE S-WATCH-QL

We extended only the SQL SELECT statement, that is used for querying data. Insertion, deletion and update are possible by standard SQL statements (even if not in a direct way). Besides, usually, the end user, in inserting, updating and deleting data, manages the clinical database by suitable graphical interfaces.[3, 12]

Functions and predicates

We call *granule* an instant value, that is (of course) associated to a granularity level. A *chronon* is an instant defined with the finest granularity available in the system (in this implementation the finest level is "seconds").[8] In tab. I are shown the functions that operate on temporal data types, i.e. instants and intervals.

Function	Description
instant .. instant → interval	Interval constructor
NOW() → *instant*	Actual time
STARTCHRON(*instant*) → *instant*	Extraction the first chronon of an instant
ENDCHRON(*instant*) → *instant*	Extraction of the last chronon of an instant
CONVGRAN@(*instant*) → *instant*	Granularity conversion
GETGRAN(*instant*) → *number*	Granularity extraction
DEGEN(*instant*) → *interval*	Degenerate interval constructor
PRED(*instant,number*) → *instant*	Predecessor
SUCC(*instant, number*) → *instant*	Successor
PRED@(*instant, number*) → *instant*	Predecessor at granularity
SUCC@(*instant, number*) → *instant*	Successor at granularity
CONVGRAN(*number, instant*) → *instant*	Granularity conversion
ELAPSED@(*interval*) → *number*	Interval duration
START(*interval*) → *instant*	Extraction of the starting element
END(*interval*) → *instant*	Extraction of the ending element
IUNION(*interval,interval*) → *interval*	Interval union
IINTERSECTION(*interval, interval*) → *interval*	Interval intersection

Tab. I. Temporal functions.

Some suitable predicate are defined for instants and for intervals. Equality (symbol "=") either between pairs of instants or between pairs of intervals requires equality of respective granularity levels. The EQUAL@ predicate checks equality between instants at a given granularity level @. We said that defining an instant *i* through a granule involves a certain indeterminacy; the entity that is related to the instant *i* is, in fact, located somewhere between STARTCHRON(*i*) and ENDCHRON(*i*). Sorting relationships between instants (e.g. $a < b$) are true if and only if they are true for every couple of chronons belonging to the operands: $a < b$, then, means ENDCHRON(*a*) < STARTCHRON(*b*). The same principle is applied to intervals. Degenerate intervals have zero duration, so the behavior of the predicates that operate over intervals consider the case of degenerate intervals and behave accordingly (for example a degenerate interval can never CONTAIN a

non degenerate interval, even if the first one is defined at a coarser granularity level).

The SELECT statement

The new SELECT statement includes the additional clauses VALID, WHEN, COALESCE ON, EXPAND BY, MOVING WINDOW, some new keywords, functions and predicates. Here's the simplified syntactical structure of the SELECT command:

```
SELECT [COALESCED] ......
[VALID ......]
[INTO ......]
FROM ......
[WHERE ......]
[WHEN ......]
[COALESCE ON ......]
[EXPAND BY ......]
[MOVING WINDOW ......]
[GROUP BY ......]
[HAVING ......]
[ORDER BY ......]
```

The result of a correct SELECT statement is always a relation. The name of the resulting relation can be specified in the INTO clause.

The SELECT clause. This clause contains a list of valid expressions, that may be simple attributes or complex expressions built on functions, predicates, attributes and constants. These expressions will compute the values of each tuple belonging to the resulting relation. Together with each expression a name may be present that will be assigned to the related attribute.

The VALID clause. The resulting relation can be historical or non historical and that depends on the presence of the VALID clause and on its arguments. Here a temporal expression (an expression that gives an instant or an interval data type result) must be present. This expression may be built on whatever time-related attribute and function, and its result defines the valid time of each tuple (the result is assigned to the TIME attribute).

The INTO clause is for defining the name of the resulting relation.

The FROM clause is used to specify the list of all the relations that will be involved by the SELECT statement.

The WHERE clause. Here can be formulated a criterion used to define the set of tuples to operate on. The criterion is in the form of a predicate; tuples for which the predicate will result TRUE will be included, other ones will be discarded. Temporal data (instants and intervals), functions and predicates cannot be present in this clause. If needed, a predicate formed on either temporal or atemporal attributes, functions and predicates, can be present in the WHEN clause.

The WHEN clause. This clause works like the WHERE clause. Here a valid predicate, made up of temporal and atemporal expressions, can be present. If a predicate p can be split into an expression such as "p_a AND p_t", where p_a is an atemporal expression and p_t is a temporal expression, the query performance will be better if p_a is placed into the WHERE clause and p_t is placed into the WHEN clause.

The COALESCE clause. Let's say that two tuples are *atemporally equal* if all the corresponding values are equal, excluding from this comparison the attribute TIME. The purpose of this clause is to coalesce into one final tuple groups of tuples that are atemporally equal and that have overlapping or meeting values of the TIME attribute. The valid time of the final tuple will be the union of all the valid times of the involved tuples.

The EXPAND BY clause. Inside this clause a specific level of granularity @ is defined. A given interval can contain a given set of instants defined with a certain level of granularity @. This clause divides a valid time into a set of intervals, each of them lasting one single instant defined at the granularity @. Every original tuple is replicated into a set of tuples atemporally equal to the original one, and time stamped with an interval corresponding to a single instant defined at the granularity @. If the valid time of an original tuple and the level of granularity @ satisfy certain conditions, the valid time of the original tuple is equal to the union of the valid times of the relative resulting tuples.

The GROUP BY clause. If we consider only a subset of the attributes that characterize two tuples belonging to the same relation, they can be considered to be equal. It is often useful to group tuples that share the same values of a subset of their attributes. By using the GROUP BY clause, such groups of tuples generate an unique tuple that can contain results coming from aggregate functions: these are special functions that operate over all the set of tuples belonging to the group, like MAX() or AVG(), to evaluate, respectively, the maximum value and the average on collections of numeric values.

The MOVING WINDOW clause. This clause defines the behavior of the moving window operator.[9] The purpose of this operator is to scan a temporally sorted relation and to group all the tuples that fall within a time interval of a given duration. For every group of tuples a single tuple is generated. This tuple can contain results that come from aggregate functions. Depending on the temporal positions of tuples and on the size of the window, every tuple may belong to one or more groups (and can contribute to the generation of different resulting tuples). To select only some of the resulting tuples, the HAVING

clause can be used. The principle we adopted is that operations must not depend on special attribute's semantics[6], such as time invariant keys[9]: we defined the construct "MOVING WINDOW ... GROUP BY <attributes>" to group tuples that have the same values for a set of attributes (to group, for example, data related to the same patient). Another useful construct is "MOVING WINDOW n TUPLES" that allows us to define a window that includes a given constant number of tuples. That wouldn't be possible by a window with a constant temporal size, if tuples are not equally temporally spaced. Through a "MOVING WINDOW 2 TUPLES" clause it is possible to build states on couples of consecutive events. The ALLTIME and ANYTIME options permit to group only tuples whose valid time is, respectively, completely or partially contained into the moving window.

The HAVING clause is used to select only a subset of the tuples resulting from a grouping operation such as GROUP BY or MOVING WINDOW. A valid predicate (that can contain aggregate functions) must be present in this clause.

The ORDER BY clause is used to define an ordered set of attributes and directions (ascending or descending) on which the resulting tuples will be sorted.

QUERYING A CLINICAL DATABASE

The relational temporal data model described above has been applied to a clinical database that comes from an anesthesia clinical record[10]. The information contained here concerns mainly actions made on the patient (induction, intubation, and so on), administration of drugs, gases and liquids, and monitoring of physiological parameters. All these data have a certain position on the temporal axis, that is very important to discover mutual relationships among events and conditions and to classify the patients according to different physiological behaviors.

Anesthetic records are usually of limited duration, event if a big amount of data is collected during an intervention, and each data element is reported as soon as it is obtained, already arranged by time on cartesian axes. In this situation it is very important to have at disposal temporal languages like S-WATCH-QL, in order to be able to identify, with ad-hoc off-line (wrt the intervention moment) queries, some different cathegories of patients, by the construction of more abstract information related to specific temporal trends of patient's data.[2]

For example, an important parameter, related to the respiratory system, often monitored during anesthesia is end-tidal CO_2 (ETCO2), that measures the percentage of CO_2 at the end of exhalation. A sudden fall in the ETCO2 may be related to cardio-circulatory problems.

The following query is one of the queries needed to discover the exit from a steady state of ETCO2. Let's say that a steady state is characterized by a time interval of ten minutes, during which the difference between the minimum and maximum values of ETCO2 is no more than 1%. We have a relation *etco2* that contains all the measures of ETCO2 pertaining to different operations. The following query constructs a relation *state* that contains tuples describing a set of steady states.

```
SELECT AVG(e.etco2) average,
  MIN(e.etco2) minimum,
  MAX(e.etco2) maximum,
  e.operation_id
VALID e.TIME, WINDOW
INTO state
FROM etco2 e
MOVING WINDOW 10 minutes
GROUP BY e.operation_id
HAVING
  MAX(e.etco2) - MIN(e.etco2) <= 1
  AND
  COUNT(e.etco2)>= 8
```

In the above query, the GROUP BY keyword belongs to the MOVING WINDOW clause and allows us to group data belonging to the same operation. The condition "COUNT(...) >= 8" avoids the user to consider temporal windows not containing a sufficient number (in this case 8) of ETCO2 measurements. The next step is to detect a sudden fall in the ETCO2. We define a sudden fall of ETCO2 as a difference equal or greater than 2% between the ETCO2 value related to a steady state (the average on the state period) and an ETCO2 measurement at the instant immediately following the end of the steady state. This is done by the following query.

```
SELECT s.average, e.etco2 new,
  START(e.TIME) fall_instant
VALID s.TIME
INTO fall
FROM state s, etco2 e
WHERE s.average-e.etco2 >= 2 AND
  s. operation_id = e.operation_id
WHEN ELAPSEDMINUTES
  (END(s.TIME)..START(e.TIME))=2
```

The above query computes a relation containing the description of only the steady states followed by a sudden fall in the ETCO2. The schema of the new historical relation *fall* is composed by the attribute identifying the steady state (s.average and the valid time s.TIME) and by the attributes identifying the sudden fall (the ETCO2 measurement after a steady

state and the temporal location of this measurement). The ELAPSEDMINUTES function must be equal to 2 because bounds are enclosed in the interval's duration.

SYSTEM ARCHITECTURE

We implemented S-WATCH-QL, by Visual C++, on PC-IBM compatible computers with MS Windows environment. The system allows the user to input queries in a textual format and to examine the resulting relations, rendered in a tabular format in separate windows. The system uses an external SQL interpreter module: by suitable drivers the system is allowed to access data stored by different database systems.[13]

It is not possible to convert a S-WATCH-QL statement into an equivalent set of SQL statements. We use the SQL interpreter mainly as a low level data access module, and for data ordering, cartesian product and atemporal selection functionalities. Evaluation of expressions, predicates, atemporal and temporal clauses requires internal processing. The activity of evaluation of a S-WATCH-QL statement, that requires a complex system architecture, follows these steps: a) syntactic analysis and checking; b) translation in an executable form; c) execution (by using also SQL interpreter); d) presentation of results.

DISCUSSION AND CONCLUSIONS

Some relevant features characterize S-WATCH-QL, in respect with other contributions in medical informatics and database literature.[1,5,9]

- Historical relations have homogeneous valid times: either *states* or *events*[5] can be mixed inside the same relation by using intervals and degenerate intervals.
- Managing of temporal attributes with varying and mixed granularity is allowed: a temporal attribute can contain a value that can be updated by a value at a different granularity level; an interval can be defined by starting and ending instants given at different granularity levels; a relation can contain different data temporally located at different granularity levels. S-WATCH-QL is not able to manage indeterminacy not related to calendar date granularities, like in [1, 3, 4].
- It is possible: a) to convert relations from non-historical to historical and vice-versa; b) to directly and simultaneously access to non-historical and historical relations; c) to compute valid times by user-defined expressions (VALID clause).
- The COALESCE, EXPAND BY and MOVING WINDOW clauses support variable granularity and degenerate intervals.
- The MOVING WINDOW CLAUSE has been extended (MOVING WINDOW ... GROUP BY, MOVING WINDOW ... TUPLES, ALLTIME AND ANYTIME OPTIONS).

Acknowledgments

This work was partially supported by contributions from: MURST Italian National Project for Medical Informatics; Department of Biomedical Engineering of the Politecnico di Milano; CNR's Centro Studi per la Teoria dei Sistemi. We thank also dr. Bruno Brunetti and dr. Stefano Fattore, for their help in focusing the clinical problem.

References

1. Das AK, Musen MA. A temporal query system for protocol directed decision support. Methods of information in medicine, 358-370, 33, 1994.
2. Shahar Y. A knowledge-based method for temporal abstraction of clinical data. Ph. D. dissertation in medical information sciences, Stanford University 1994.
3. Combi C, Pinciroli F, Musazzi G, Ponti C. Managing and displaying different time granularities of clinical information. In: Ozbolt, JG (cd.), 18. Symposium on Computer Applications in Medical Care. Hanley & Belfus. Philadelphia 1994; 954 - 958.
4. Combi C, Pinciroli F, Cavallaro M, Cucchi G. Querying temporal clinical databases with different time granularities: the GCH-OSQL language. Symposium on Computer Applications in Medical Care. 19, Philadelphia, Hanley & Belfus, 1995; 326 - 330.
5. Sarda NL. HSQL: a Historical Query Language. in 12. 110-138.
6. Snodgrass RT. The TSQL2 temporal query language. The TSQL2 Language Design Committee Kluwer Academic Publishers, Boston, 1995.
7. Date CJ. A guide to the SQL standard. Addison Wesley 1989.
8. Jensen CS, Clifford J, Elmasri R, Gadia SK et al. A consensus glossary of temporal database concepts SIGMOD Record, vol. 23, 52-64,1994.
9. Navathe SB, Ahmed ER. Temporal extensions to the relational model and SQL in 12. 92-109.
10. Barash PG, Cullen BF, Stoelting RK. Clinical anesthesia. J.B. Lippincott Company 1992.
11. Tansel AU, Clifford J, Gadia S, Jajodia S, Segev A, Snodgrass R. Temporal databases: theory, design and implementation. Benjamin Cummings, Redwood City, CA, 1993.
12. Cousins S, Kahn M, Frisse M. The display and manipulation of temporal information. SCAMC 1989, 76-80.
13. Kruglinski DJ. The Visual C++ manual. McGraw Hill, New York, 1994 (in Italian).